Appendix D: KidCode® Lingo Client/Server Rebus Message Handling Component Scripts


```
--- Main Movie Scripts
----- Sprite Assignments
--
-- sprite 3              reserved for the ViewSymbols buttons
-- sprite 4              reserved for the ViewGuesses buttons
-- sprite 5              reserved for the NewTemplate buttons
-- sprites 45 thru 48    buttons for scrolling symbol palette

---- Sprites below are assigned to cast members dynamically
--
-- sprites 15 thru 20    reserved for symbols in symbol palette
-- sprites 21 thru 35    reserved for symbols in Rebus message
-- sprites 39 thru 44    reserved for typein text guesses
```

```
on startMovie
  -- Public Globals
  global rmG_registeredUsers        -- list of KidCode system usernames
  global rmG_userName               -- records username
  global rmG_mode                   -- #display, #display_noReply, or
#author
  global rmG_mailData               -- the active message including
header
  global rmG_messageNumber
  global rmG_testState              -- 0,1,2,3  template set for user
tests
  global rmG_userGroup

  -- private internal globals
  global rmG_traceFlag      -- for debugging

  global rmG_symbolcastName -- name of cast library of symbols to use
  global rmG_templates      --- list of template names for this user
group
  global rmG_theTemplateIndex   --- index of the current template
  global rmG_symbolGroup    --- lists symbol cast members for current
template

  global rmG_msgBody         -- the active messageBody
  global rmG_state           -- #decIn, #codIn, #new, or #gotIt
  global rmG_guesses        -- list of property lists with info on
guesses

  set rmG_traceFlag = 0     -- set to 1 to turn on tracing, else 0

  set the fileName of castLib "Templates" to the pathName &
"Templates.cst"
  set the fileName of castLib "Symbols" to the pathName & "Symbols.cst"

  -- NEXT EXISTS SO THAT REBUS MOVIE CAN BE RUN IN SIMULATION MODE
  -- IT IS SET TO TRUE BY THE INITWINDOW FUNCTION WHEN RUN AS
  -- EMAIL COMPONENT!!!!
```

```
  global rmG_noSimulate
  --  set rmG_noSimulate = TRUE

  if rmG_noSimulate then  -- all globals initialized in initWindow
    tell the stage to emh_continue(#msgHandler)

  else   -- SIMULATE MODE
    initSimulation

  end if -- SIMULATE MODE

end

-- When email main opens the Rebus movie this function is called.
-- The openwindow function cannot accept an argument.
-- Therefore initwindow below is necessary to pass the initial
-- arguments to Rebus.

on openWindow
  -- put "EXIT OpenWindow" & " in frame" & the frame
end openWindow

---------------------------------------------------------------------

on initSimulation
  -- Public Globals
  global rmG_registeredUsers        -- list of KidCode system usernames
  global rmG_userName               -- records username
  global rmG_mode                   -- #display, #display_noReply, or
#author
  global rmG_mailData               -- the active message including
header
  global rmG_messageNumber
  global rmG_testState              -- 0,1,2,3  template set for user
tests

  --- internal globals
  global rmG_symbolcastName      -- which castlib to use
  global rmG_symbolGroup   --- lists cast members in currently selected
group
  global rmG_templates     --- property list with entry for each
template
  global rmG_theTemplateIndex   --- index of the current template
  global rmG_msgBody
  global rmG_state
  global rmG_guesses

   -- Initialize variables that would have been passed by email main
  set rmG_userName = "user1"
  set rmG_registeredUsers = ["user2", "user1"]
  set rmG_mode = #author
  set rmG_mailData = [#to:"User1", #from:" ", #re:"Rebus Challenge",¬
                      #mimeType:"Rebus", #status:"new"]
  addProp(rmG_mailData, #date, the abbreviated date)
```

```
107     addProp(rmG_mailData, #msgBody, [#new, [], "default"])
108     set rmG_messageNumber= 1
109     set rmG_testState = 2     -- user group; determines castLib and
110   templates
111
112     -- Initialize private globals
113     set rmG_msgBody = getProp(rmG_mailData, #msgBody)  -- set up pointer
114     set rmG_state = getAt(rmG_msgBody, 1)
115
116     initializeTemplates       -- initializes rmG_templates
117
118     set rmG_theTemplateIndex = getPos(rmG_templates, "default")
119     set theTemplate = "default"
120     set rmG_symbolcastName = getTemplateProp(theTemplate, #library)
121     set rmG_symbolGroup = getTemplateProp(theTemplate, #symbols)
122
123     set rmG_guesses = []  -- guesses initialized after template is
124   selected
125
126     --- Format the font properties of text fields and the MessageSpace
127     setUpMessageSpace()
128     formatFields()
129
130     --- allow To field to be a listbox
131     puppetSprite 50, TRUE
132
133
134     fillToList()
135
136     -- data structures to improve efficiency in text and graphics layout
137     global rmG_layoutIndex
138     set rmG_layoutIndex = [:]  -- records word position and loc info by
139   index
140
141   end initSimulation
142
143   ------------------------------------------------------------------------
144   --- closeWindow is not called unless Rebus plays as a MIAW.
145
146   on closeWindow
147     finishMovie
148   end closeWindow
149
150   ------------------------------------------------------------------------
151   -- stopMovie is not called if Rebus plays as a MIAW
152
153   on stopMovie
154     finishMovie
155   end
156
157   ------------------------------------------------------------------------
158   -- This needs to play whenever the movie closes,
159   -- whether as MIAW or standalone.
```

```
160
161   on finishMovie
162     global rmG_noSimulate
163
164     clearHdrFields
165     clearMessageSpace
166     set the member of sprite 3 = member "SymbolButtonUp"
167     set the member of sprite 4 = member "GuessesButtonUp"
168     if soundBusy(1) then sound stop 1
169
170     -- next line will cause problems for the main movie
171     -- for now just reset this by hand if you have been running in
172     -- email mode and want to switch to simulate mode.
173     -- Note, unless Director is restarted, the rmG_noSimulate global
174     -- stays set even when a new Rebus movie is loaded.
175     -- set rmG_noSimulate = FALSE
176
177   end
178
179
180   --- Template handlers
181   --- All information needed for each Template is stored
182   --- in the "Templates" castLib.
183   --- Each template is stored as a list with the following
184   --- elements:
185   ---   template text string
186   ---   list of indices of coded words
187   ---   text string name of castlib for template's symbols
188   ---   list of member numbers in castlib for template's symbols
189   ---
190   --- e.g. [ "Can a truck fly?", [3,4], "UTsymbols",[4,23,24,25]]
191
192   ------------------------------------------------------------------
193   --- initializeTemplates
194   --- set up the rmG_templates list of Rebus template names for the
195   --- user group determined by rmG_testState
196
197   on initializeTemplates
198     global rmG_testState      --- user testing group
199     global rmG_templates      --- list of template names for this usergroup
200     global rmG_userGroup
201
202     set rmG_templates = ["default"]   -- reset for safety
203
204     if rmG_testState = 0 then   -- administrator
205       repeat with str in ["girlrain", "clownsaid", "withoutsun"]
206         add(rmG_templates, str)
207       end repeat
208
209     else   --- user group
210
211       -- Sentences common to all groups
```

```
212      if 0 then --- sentences have been split up among pairs (1,2) and
213  (3,4)
214        repeat with str in ["attrib2", "attrib3", "attrib5", ¬
215                            "quant1", "quant5", "quant6", ¬
216                            "verb1", "verb3", "verb4"]
217          add(rmG_templates, str)
218        end repeat
219      end if
220
221      -- Add specialized sentences
222
223      case rmG_testState of
224        1:  -- attrib: 2,3  quant: 3a,4a,6   verb: 1,3,5a
225          repeat with str in ["verb5a", "attrib2", "quant6", "attrib3",¬
226                              "verb1", "verb3",  ¬
227                              "quant4a", ¬
228                                "quant3a"]
229            add(rmG_templates, str)
230          end repeat
231
232        2: --attrib: 4b,5  quant: 1,2,3b,5   verb: 2b,4,5b
233          repeat with str in ["attrib4b", "quant3b", "verb4", ¬
234                              "quant2", "verb2b", "attrib5",¬
235                              "quant1", "quant5", "verb5b"]
236            add(rmG_templates, str)
237          end repeat
238
239        3: --attrib: 2,3  quant: 3a,4b,6   verb: 1,3,5c
240          repeat with str in ["verb5c", "attrib2", "quant6", "attrib3",¬
241                              "verb1", "verb3",  ¬
242                              "quant4b", ¬
243                                "quant3a"]
244            add(rmG_templates, str)
245          end repeat
246
247        4: --attrib: 4b,5  quant: 1,2,3c,5   verb: 2b,4,5b
248          repeat with str in ["attrib4b", "quant3c", "verb4", ¬
249                              "quant2", "verb2b", "attrib5",¬
250                              "quant1", "quant5", "verb5b"]
251            add(rmG_templates, str)
252          end repeat
253
254        otherwise:
255          alert "ERROR:initializeTemplates invalid user group."
256
257      end case
258
259    end if   -- administrator or usergroup
260
261  end initializeTemplates
262
263
264
```

```
265  -- Accessor functions for template info:
266
267  on getTemplateProp memberName, prop
268
269    set cNum = the number of member memberName of castLib "Templates"
270
271    if cNum < 1 then
272      alert "ERROR:no member " & memberName & " in castlib Templates"
273      return(0)
274    end if
275
276    case prop of
277      #text: return(getAt(value(the text of field cNum), 1))
278      #coded: return(getAt(value(the text of field cNum), 2))
279      #library: return(getAt(value(the text of field cNum), 3))
280      #symbols:return(getAt(value(the text of field cNum), 4))
281      otherwise: return(0)
282    end case
283
284  end getTemplateProp
285
286  ------------------------------------------------------------------------
287  ------------------------------------------
288  --- Returns a partial message body structure from a simple
289  --- list of the form [templateString, list_of_code_word_indices]
290  --- Used to read from templates cast members into message bodies
291
292  on readTemplate allwords, indices
293
294    set template = []
295    set val = [:]
296    set nwords = the number of words in allwords
297    set str = ""
298    set charPos = 1
299    set j = 1
300    set i = 1
301
302    repeat while j <= nwords
303      repeat while not getOne(indices, j) and j <= nwords
304        set str = str & word j of allwords & " "
305        set j = j+1
306      end repeat
307      if length(str) > 0 then  -- add uncoded words
308        addProp(val, #text, str)
309        addProp(val, #code, 0)
310        addProp(val, #charPos, charPos)
311        addProp(val, #wordIndex, i)   -- index of first word
312        add(template, duplicate(val))
313        set val = [:]
314        set charPos = charPos + length(str)
315        set str = ""
316      end if
317      if j <= nwords then  --- add the coded word
```

```
318         addProp(val, #text, word j of allwords & " ")
319         addProp(val, #code, 1)
320         addProp(val, #charPos, charPos)
321         addProp(val, #wordIndex, j)
322         add(template, duplicate(val))
323         set val = [:]
324         set charPos = charPos + length(word j of allwords)+ 1
325         set j = j + 1
326       end if
327
328       set i = j
329
330     end repeat
331
332     return(template)
333   end readtemplate
334
335   --- Message handlers
336
337   ----------------------------------------------------------------
338   -- openMessage function is only called by other Rebus functions
339   -- It does not handle laying out text properly for messages that
340   -- may have been in alternative mailboxes.  See msh_openMessage.
341
342   on openMessage
343     global rmG_msgBody, rmG_mailData, rmG_mode, rmG_state, rmG_symbolGroup
344     global rmG_lineHeightFix
345     global rmG_noSimulate      --- False to run simulation independent of
346   email
347     global rmG_testState
348     global rmG_symbolcastName
349     global rmG_userName
350
351     global rmG_traceFlag
352
353     put "In REBUS:openMessage"
354
355     if rmG_noSimulate then
356       set rmG_msgBody = getProp(rmG_mailData, #msgbody)
357
358       if count(rmG_msgBody) = 0 then    -- start a new message
359
360         startMessage("default")
361         --- Fill header fields
362         put getProp(rmG_mailData, #to) into field "To"
363         put getProp(rmG_mailData, #from) into field "From"
364         put the abbreviated date into field "Date"
365         put "Rebus Challenge" into field "Re"
366
367       else  --- Handle existing message
368         -- put "OPEN rmG_msgBody" & rmG_msgBody
369
370         set rmG_state = getAt(rmG_msgBody, 1)
```

```
371         set whichTemplate = getAt(rmG_msgBody,3)
372
373         clearMessageSpace
374         putHdrFields()    -- read headers into field members
375
376         --- setup symbols in graphics palette to match template
377         set rmG_symbolGroup = getTemplateProp(whichTemplate, #symbols)
378         setupSymbolPalette rmG_symbolcastName
379         initializeGuesses     -- uses global var rmG_msgBody
380
381         case rmG_state of
382           #new:
383             layoutText(getat(rmG_msgBody,2), 6)  -- coded words red
384             makeLayoutIndex(getAt(rmG_msgBody, 2))
385             placeGraphics()
386
387           #decIn:
388
389             layoutText(getat(rmG_msgBody,2), 0) -- coded words white
390             makeLayoutIndex(getAt(rmG_msgBody, 2))
391             placeGraphics()
392
393           #codIn:
394
395             layoutText(getat(rmG_msgBody,2), 6) -- coded words red
396             makeLayoutIndex(getAt(rmG_msgBody, 2))
397             placeGraphics()
398
399           #gotIt:
400             -- hide scrolling symbol palette
401             repeat with i = 15 to 20
402               -- puppetsprite i, TRUE
403               set the visible of sprite i = FALSE
404             end repeat
405             layoutText(getat(rmG_msgBody,2), 6)  -- coded words white
406             makeLayoutIndex(getAt(rmG_msgBody, 2))
407             placeGraphics()
408             go to "gotIt"
409         end case
410
411       end if
412
413
414     else   -- Simulate mode
415
416       setupSymbolPalette rmG_symbolcastName
417       set rmG_state = getAt(rmG_msgBody, 1)
418       if rmG_state = #new then      -- user picked a template
419         set rmG_msgBody = newMessageBody(whichTemplate)
420         layoutText(getAt(rmG_msgBody, 2), 6)   -- 6 is white
421         makeLayoutIndex(getAt(rmG_msgBody, 2))
422         initializeGuesses     -- uses global var rmG_msgBody
423       else   --- don't want to change message body or index
```

```
424         layoutText(getAt(rmG_msgBody, 2), 0)    -- 0 is white
425         makeLayoutIndex(getAt(rmG_msgBody, 2))
426         placeGraphics()
427         if rmG_state = #gotIt then
428           go to "gotIt"
429         end if
430       end if   -- Simulate
431
432     end if
433
434     if rmG_mode = #display then
435       set the editable of member "To" = FALSE
436     else if rmG_mode = #author then
437       fillToList()
438     end if
439
440
441   end openMessage
442
443
444
445   --- Start message gets called only when there is a new template
446   --- If a previous msg existed it is abandoned.
447
448   on startMessage whichTemplate
449     global rmG_msgBody, rmG_state, rmG_symbolGroup, rmG_symbolcastName
450
451     clearMessageSpace
452     set rmG_msgBody = newMessageBody(whichTemplate)
453     set rmG_state = getAt(rmG_msgBody, 1)
454     layoutText(getat(rmG_msgBody,2), 6)  -- coded words red
455     makeLayoutIndex(getAt(rmG_msgBody, 2))
456     set rmG_symbolGroup = getTemplateProp(whichTemplate, #symbols)
457     setupSymbolPalette rmG_symbolcastName
458
459     initializeGuesses     -- uses global var rmG_msgBody
460
461   end startMessage
462
463
464
465   --- function for coder to reply with a gotIt message
466
467   on handleGotIt
468     global rmG_state, rmG_msgBody, rmG_mailData, rmG_mode
469
470     if rmG_mode = #display then
471       alert "You must click on reply first!"
472       return(0)
473     end if
474
475     if rmG_state <> #decIn then
476       set rmG_state = #gotIt
```

```
477       deleteAt(rmG_msgBody,1)
478       AddAt(rmG_msgBody,1, #gotIt)
479       put "Rebus Success!" into field "Re"
480       setProp(rmG_mailData, #re, "Rebus Success!")
481       alert "click on send to send Success!"
482
483     else alert "Only the coder can tell you GOT IT!"
484
485   end handleGotIt
486
487
488
489
490   -- Make new messageBody data structure from a template by adding
491   -- the symbol and the guess to coded words.
492
493   on newMessageBody  whichTemplate
494
495     set mBody = readTemplate(getTemplateProp(whichTemplate, #text),¬
496                              getTemplateProp(whichTemplate, #coded))
497
498     repeat with i = 1 to count(mBody)
499       set nextItem = getAt(mBody,i)
500       set nextText = getProp(nextItem, #text)
501       if getProp(nextItem, #code) then
502         deleteAt mBody, i
503         AddProp nextItem, #symbol, 0
504         AddProp nextItem, #guess, " "
505         addAt mBody, i, nextItem
506       end if
507     end repeat
508
509     -- add the rebusState as first item of messageBody
510     -- and the template name as the last item of messageBody
511     return list(#new, mBody, whichTemplate)
512
513   end newMessageBody
514
515   ------------------------------------------------------------------------
516   --- update the character positions and
517   --- read text strings into the message body
518   --- for now assume guesses are updated elsewhere
519
520   on updateMsgBody
521     global rmG_msgBody
522
523     set msg = getAt(rmG_msgBody,2)
524     set nItems = count(msg)
525     set i = 1
526
527     if 0 then
528
529       repeat with x in msg
```

```
530         set index = getProp(x, #wordIndex)
531         set charPos = getLayoutProp(index, #firstCharPos)
532         -- setProp(x, #charPos, charPos)
533         set symbol = getLayoutProp(index, #symbolSprite)
534         if symbol then
535           setProp(x, #symbol, the memberNum of sprite symbol)
536         end if
537         if i < nItems then
538           -- set nextIndex = getProp(getAt(msg, i+1), #wordIndex)
539           set lastChar = getLayoutProp(getProp(getAt(msg, i+1),
540   #wordIndex),¬
541                                        #firstCharPos) -1
542         else
543           set lastChar = the length of field "MessageSpace"
544         end if
545         set ts = char charPos to lastChar of field "MessageSpace"
546         setProp(x, #text, ts)
547         set i = i+1
548       end repeat
549     end if
550   end updateMsgBody
551   ------------------------------------------------------------------------
552   ---
553   --  CLEAR THE TO, FROM, RE,. DATE, MIMETYPE FIELDS
554
555   on clearHdrFields
556     put " " into field "To"
557     put " " into field "ToDown"
558     put " " into field "From"
559     put " " into field "Re"
560     put " " into field "Date"
561   end
562
563
564   ------------------------------------------------------------------------
565   --
566   -- updateHeader reads info from the message header
567   -- fields back into the message
568
569   on updateHeader
570     global rmG_mailData
571
572     setProp rmG_mailData, #to, the text of member "To"
573     setProp rmG_mailData, #from, the text of member "From"
574     setProp rmG_mailData, #date, the text of member "Date"
575     setProp rmG_mailData, #re, the text of member "Re"
576     setProp rmG_mailData, #mimetype, "rebus"
577
578   end updateHeader
579
580   ------------------------------------------------------------------------
581
582   on putHdrFields
```

```
583     global rmG_mailData
584
585     put getProp(rmG_mailData, #to) into field "To"
586     put getProp(rmG_mailData, #from) into field "From"
587     put getProp(rmG_mailData, #re) into field "Re"
588     put getProp(rmG_mailData, #date) into field "Date"
589
590   end putHdrFields
591
592   -- Symbol sprites
593   -- getFreeSymbolSprite returns the sprite number of a
594   -- free sprite if it exists.  Otherwise it returns 0.
595   -- If a sprite reserved for a symbol has an empty
596   -- castmember then it is available for use.
597   -- This way we avoid using a global list of free sprites.
598
599   on getFreeSymbolSprite
600
601     repeat with i = 21 to 35      -- sprites 21-35 reserved for symbols
602       if the memberNum of sprite i = 0 then return i
603     end repeat
604
605     return(0)  -- none found
606
607   end getFreeSymbolSprite
608
609
610
611   --- Initializes a new symbol and returns the sprite number.
612   --- If no symbol sprites are available it returns 0.
613   --- The argument graphicMember is the member, not the
614   --- memberNumber.
615
616   --- Maybe should make symbols uneditable when they are
617   --- created (if #decIn or #display) instead of in the
618   --- placeGraphics handler, as is currently the case.
619
620   on newMessageSymbol  graphicMember
621
622     set newSymbolNum = getFreeSymbolSprite()
623     if newSymbolNum = 0 then
624       beep
625       return(0)
626     else
627       puppetsprite newSymbolNum, TRUE
628       set baseloc = the loc of sprite newSymbolNum
629       set the member of sprite newSymbolNum = graphicMember
630       set the visible of sprite newSymbolNum = TRUE
631       set the moveablesprite of sprite newSymbolNum = TRUE
632       set wordIndex = 0
633       return(newSymbolNum)
634     end if
635
```

```
end newMessageSymbol

-- Palette of graphic symbols
-----------------------------------------------------------------------
-- Setup scrolling symbol palette
-- Sprites 15 thru 20 are reserved for symbols visible on the palette.
-- Symbol bitmaps are max of 100 pixels in either direction.
-- Set up one sprite directly above palette and one sprite
-- directly below the palette.

on setupSymbolPalette whichCast
  global rmG_symbolGroup  -- cast members of selected symbol group
  global rmG_mode, rmG_state

  set firstY = -55   -- sprite directly above palette

  set nextMember = 1
  repeat with i = 15 to 20
    puppetsprite i, TRUE
    set the visible of sprite i to TRUE
    if rmG_state = #decIn OR rmG_mode = #display then
      set the moveableSprite of sprite i = FALSE
    else
      set the moveableSprite of sprite i = TRUE
    end if
    -- set the ink of sprite i to 36  -- background transparent
    set firstSymbolY = firstSymbolY + 120
    set the member of sprite i = member getAt(rmG_symbolGroup,
nextMember) of castLib whichCast
    set the memberIndex of sprite i = i-14   --- index in symbolgroup
    set the locH of sprite i to 582
    set the locV of sprite i to firstSymbolY
    set firstY = firstY + 120

    -- sprite should have palette behaviors
    set the paletteLoc of sprite i = the loc of sprite i

    set nextMember =  (nextMember mod 6) + 1
  end repeat

  updatestage
end setupSymbolPalette

-------------------------------------------------------------------------

--- sprites 15 thru 20 are reserved for symbols
--- on scrolling palette
--- whichCast is a string that refers to the castName in the
--- templates castLib.

on chooseCast whichCast
  global rmG_symbolGroup
  global rmG_symbolcastName
```

```
689
690     set rmG_symbolGroup = getTemplateProp(whichCast, #symbols)
691     set rmG_symbolcastName = getTemplateProp(whichCast, #library)
692     setupSymbolPalette rmG_symbolcastName
693
694
695   end chooseCast
696
697   -- Palette Symbol
698   -- Intellinet Inc. behavior
699   -- Implements behaviors for a palette sprite.
700   -- This includes the ability to create copies of itself
701   -- if placed in a message and the ability to scroll.
702
703
704   property paletteloc, memberIndex
705
706   -- paletteLoc is the location of the sprite in the palette.
707   -- memberIndex is the Index of the sprites castMember in the list
708   -- of castMembers in the symbol palette. (rmG_symbolGroup)
709
710   on mouseUp me
711     global rmG_symbolcastName, rmG_state, rmG_mode
712
713     set upLoc = point(the mouseH, the mouseV)    -- the clickloc???
714
715     if rmG_state <> #decIn and rmG_mode <> #display then -- message can be
716   edited
717       -- put "symbol " & the spriteNum of me & " uploc = " & uploc
718
719       set textCast = the number of member "MessageSpace"
720       set textSprite = 2    --reserved for the message space
721       set castNum = the memberNum of sprite the clickon
722
723       -- it's an original from the symbol palette put it back
724       set the loc of sprite the spriteNum of me to paletteLoc
725
726       if inside(upLoc, the rect of sprite textSprite) then
727
728         set wordI = locToWordPos(member textCast, ¬
729                                  upLoc - the loc of sprite textSprite)
730         if wordI > 0 then
731           if codewordP(wordI) then
732             if getLayoutProp(wordI, #symbolSprite) then -- word already
733   has a symbol
734               --- just swap its cast member with the new one
735               set the memberNum of sprite getLayoutProp(wordI,
736   #symbolSprite) = ¬
737                        the memberNum of sprite the spriteNum of me
738               updatestage
739             else
740               set newSpriteNum = newMessageSymbol(member castNum of
741   castLib¬
```

```
742                                                          rmG_symbolcastName)
743             if newSpriteNum <> 0 then
744               placeSymbolInText(sprite newSpriteNum, wordI)
745             else -- can't do anything, no more sprites.
746             end if
747
748           end if
749         end if
750       end if
751     else nothing --- dragged somewhere outside of MessageSpace
752
753     updatestage
754
755   end if    --- symbols editable?
756 end mouseUp
757
758 --- Palette Symbol continued
759
760 --- to scrollUp
761 --- cycle in the next CastMember in the currentSymbolGroup
762
763 on scrollUp me
764   global rmG_symbolGroup, rmG_symbolcastName
765
766   if memberIndex = count(rmG_symbolGroup) then
767     set memberIndex = 1
768   else set memberIndex = memberIndex + 1
769
770   set the member of sprite the spriteNum of me = member ¬
771               getAt(rmG_symbolGroup, memberIndex) of castLib
772 rmG_symbolcastName
773
774 end scrollUp
775
776
777
778 --- to scrollDown
779 --- cycle in the previous CastMember in the currentSymbolGroup
780
781 on scrollDown me
782   global rmG_symbolGroup, rmG_symbolcastName
783
784   if memberIndex = 1 then
785     set memberIndex = count(rmG_symbolGroup)
786   else set memberIndex = memberIndex - 1
787
788   set the member of sprite the spriteNum of me = member ¬
789               getAt(rmG_symbolGroup, memberIndex) of castLib
790 rmG_symbolcastName
791
792 end scrollDown
793
794
```

```
795  -- Message Symbol
796  -- Intellinet Inc. behavior
797  -- Implements behaviors for a graphic sprite in message.
798  -- When the message has a text component, the message symbol sprites
799  -- snap to coded words.
800
801  property baseLoc, wordIndex
802
803  -- wordIndex is the index of the message word that this sprite
804  -- is attached to.
805  -- wordIndex = 0 if sprite is not attached to a coded word.
806
807
808  -- to mouseUp need to add method to allow symbols to be placed into
809  empty
810  -- messages (e.g. default template = clear) and moved around in the
811  -- messages.  Need also to add method to handle messages that consist of
812  -- nothing but symbols.  Probably check to see if template is default.
813
814
815
816  -- ReInitializes the properties of sprite and resets the
817  -- memberNumber to 0, making the sprite available for reuse.
818
819  on recycleMessageSymbol me
820
821    set wordIndex = 0
822    set baseloc = point(-100, -100)   -- offstage
823    set the visible of sprite the spriteNum of me = 0
824    set the memberNum of sprite the spriteNum of me = 0
825    set the loc of sprite the spriteNum of me = baseloc
826
827  end recycleMessageSymbol
828
829
830
831  on mouseUp me
832    global rmG_state, rmG_mode
833
834    set upLoc = point(the mouseH, the mouseV)   -- the clickloc???
835
836    if rmG_state = #decIn OR rmG_mode = #display then   --- disable symbol
837      return(0)
838    end if
839
840    --- symbol enabled
841
842    set textSprite = 2   -- reserved for the message space
843
844    if not inside(uploc, the rect of sprite textsprite) then
845      deleteSymbolInMsg(me) -- take it out of msg
846      return(0)
847    end if
```

```
848
849     -- since symbol can only be in the message
850     -- symbol was clicked in the message; maybe moved
851
852     if wordIndex <> 0  then -- moved from a coded word in message
853
854       set toWord = locToWordPos(member "MessageSpace", ¬
855                                 upLoc - the loc of sprite textsprite)
856
857       if toWord = wordIndex then -- they left sprite where it started
858         return(0)                --  do nothing
859       end if
860
861       if codeWordP(toWord) then  -- symbol moved to a different code word
862
863         moveSymbolInMsg(me, toWord)
864
865       else   -- symbol not placed on a coded word; disappear it
866         -- change this to accomodate picture messages
867
868         deleteSymbolInMsg(me) -- take it out of msg
869
870       end if
871
872     else -- symbol moved from somewhere else; maybe new or no text in msg
873       --- this shouldn't happen until we accomodate picture messages
874
875     end if
876
877     updatestage
878
879   end mouseUp
880
881
882
883   -- WordI is the index of the word in the message.
884   -- Normally wordI points to a coded word.
885   -- Checks should have been done prior to call to ensure that wordI
886   -- does not already have a symbol.
887
888   -- Note all locations in layoutIndex are relative to the origin of
889   -- the "MessageSpace" field
890
891   on placeSymbolInText  me,  wordI
892
893     global rmG_spaceWidth
894     global rmG_msgBody
895
896     set wordIndex = wordI   -- update my property
897     set fieldNum = the number of member "MessageSpace"
898     set textOrigin = the loc of sprite 2   -- sprite for "MessageSpace"
899     set message = getAt(rmG_msgBody,2)
900
```

```
901     set startChar = getLayoutProp(wordIndex, #firstCharPos)
902     set startloc = getLayoutProp(wordIndex, #firstCharLoc)
903     set endloc = getLayoutProp(wordIndex, #lastCharLoc)
904
905     If wordSpaceH(wordI) < 120 then  --includes spaces before & after
906       set shiftDistance = integer((120 - (getAt(endloc,1) - ¬
907                                           getAt(startloc,1)) ) /2)
908       --- shift at beginning of word
909       set nchars = shiftTextRight(startChar, shiftDistance, fieldNum,
910   rmG_spaceWidth)
911       set startChar = startChar + nchars
912       set startLoc = myCharPosToLoc(member "MessageSpace", startChar)
913       set endChar = getLayoutProp(wordIndex, #lastCharPos) + nchars
914
915       --- shift at the end of word
916       shiftTextRight(endChar, shiftDistance, fieldNum, rmG_spaceWidth)
917       set endLoc = myCharPosToLoc(member "MessageSpace", endChar)
918
919       setLayoutProp(wordI, #firstCharPos, startChar)
920       setLayoutProp(wordI, #firstCharLoc, startLoc)
921       setLayoutProp(wordI, #lastCharPos, endChar)
922       setLayoutProp(wordI, #lastCharLoc, endLoc)
923
924
925       -- update all forward words info and symbols
926       repeat with x in message
927         set i = getProp(x, #wordIndex)
928         if i > wordI then
929           set startchar = getLayoutProp(i, #firstCharPos) + 2*nchars
930           --- setProp(x, #charPos, startChar)  -- never change this
931           setLayoutProp(i, #firstCharPos, startChar)
932           set startLoc = myCharPosToLoc(member fieldNum, startChar)
933           setLayoutProp(i, #firstCharLoc, startLoc)
934           set endChar = getLayoutProp(i, #lastCharPos) + 2*nchars
935           setLayoutProp(i, #lastCharPos, endChar)
936           set endLoc = myCharPosToLoc(member fieldNum, endChar)
937           setLayoutProp(i, #lastCharLoc, endLoc)
938
939           --- place the symbol
940           set nextSymbol = getLayoutProp(i, #symbolSprite)
941           if nextSymbol then
942             set the baseLoc of sprite nextSymbol = ¬
943              centerOnWord(member "MessageSpace", startLoc, endLoc) +
944   textOrigin
945             set the loc of sprite nextSymbol = the baseLoc of sprite
946   nextSymbol
947           end if
948
949         end if   -- i > wordIndex
950       end repeat  -- all forward words
951
952     end if        --- need to shift words
953
```

```
954     --- handle this word's symbol placement  (whether or not text shifted)
955     setLayoutProp(wordI, #symbolSprite, the spriteNum of me)
956     set the forecolor of word wordindex of field "MessageSpace" = 0
957
958     --- update my sprite property
959     --- location is adjusted to be relative to stage origin
960     set baseLoc = centerOnWord(member "MessageSpace", ¬
961                         getLayoutProp(wordI, #firstCharLoc), ¬
962                         getLayoutProp(wordI, #lastCharLoc)) + textOrigin
963
964     set the loc of sprite the spriteNum of me = baseloc
965     --- record the symbol's castmember into the MessageBody
966     setProp(getAt(message, getLayoutProp(wordI, #msgIndex)), ¬
967                 #symbol, the memberNum of sprite the spriteNum of me)
968
969   end placeSymbolInText
970
971
972   ------------------------------------------------------
973   --- Moves the location of the symbol in the message
974   --- If necesary, checks that toWord is a codeWord should be done
975   --- prior to calling this handler.
976
977   on moveSymbolInMsg  me,  toWord
978
979     --- remove sprite from old word
980     setLayoutProp(wordIndex, #symbolSprite, 0)
981     setProp(getAt(getAt(rmG_msgBody, 2), getLayoutProp(wordIndex,
982   #msgIndex)),¬
983             #symbol, 0)
984     -- since decoder can't move symbols in message, this is the coder
985     set the forecolor of word wordIndex of field "MessageSpace" = 6  --red
986
987     --- put sprite on new word
988     placeSymbolInText(me, toWord)
989
990   end moveSymbolInMsg
991
992
993
994   --- Currently does not rejust text to create less space
995   --- around the word the symbol was moved from.
996
997   on deleteSymbolInMsg  me
998     global rmG_msgBody
999
1000    setLayoutProp(wordIndex, #symbolSprite, 0)
1001    setProp(getAt(getAt(rmG_msgBody, 2), getLayoutProp(wordIndex,
1002  #msgIndex)),¬
1003            #symbol, 0)
1004
1005    -- since decoder can't move symbols in message, this is the coder
1006    set the forecolor of word wordIndex of field "MessageSpace" = 6  --red
```

```
1007
1008      recycleMessageSymbol(me)
1009
1010    end deleteSymbolInMsg
1011
1012
1013
1014    on toggleVisible me
1015
1016      set the visible of me = not the visible of me
1017
1018    end togglevisible
1019
1020    -- message authoring handlers
1021    -- clearMessageSpace clears the workspace, the symbols and
1022    -- the guesses.
1023    -- For now, in order to simulate SENDING a MSG this function
1024    -- does not get rid of the active message in rmG_msgBody
1025
1026    on clearMessageSpace
1027      global gFreeSpriteList
1028      global rmG_noSimulate
1029
1030      if not rmG_noSimulate then
1031
1032      end if
1033
1034      --  first, make symbols invisible and free sprites
1035      repeat with i = 21 to 35
1036        recycleMessageSymbol(sprite i)
1037      end repeat
1038
1039      -- clear the references to symbol sprites from the message Index
1040      clearLayoutSymbols()
1041
1042      --  second, clear away TypeInText fields for guesses
1043      repeat with i = 39 to 44
1044        set the visible of sprite i = FALSE
1045        set the loc of sprite i = point(-100, -100)  --put offstage
1046        -- put "  " into field (the memberNum of sprite i)
1047      end repeat
1048
1049      --  third, clear away old template if any
1050      put "  " into member "MessageSpace"
1051      set the forecolor of member "MessageSpace" = 0   -- white
1052
1053      updateStage
1054
1055    end clearMessage
1056
1057    ---------------------------------------------------------------
1058    -- message authoring handlers continued
1059
```

```
1060  --- PlaceGraphics is called to put the symbols and guesses into the
1061  text.
1062  --- Assumes that the text has been laid out and the layout
1063  --- index has been created.
1064
1065  on placeGraphics
1066
1067    global rmG_msgBody
1068    global rmG_guesses, rmG_state, rmG_mode
1069    global rmG_symbolcastName
1070
1071    set textSprite = 8   -- sprite reserved for Rebus message field
1072    set textOrigin = the loc of sprite textSprite
1073    set message = getAt(rmG_msgBody,2)
1074
1075    -- First, place the symbol graphics
1076    repeat with x in message
1077      if getProp(x, #code) then          -- code word might have symbol
1078        set index = getProp(x, #wordIndex)
1079        set xMember = getProp(x, #symbol)
1080
1081        if xMember > 0 then               -- code word does have symbol
1082          set xSprite = getLayoutProp(index, #symbolSprite)
1083          if xSprite = 0 then             -- no sprite assigned to symbol
1084            set xSprite = newMessageSymbol(member xMember of castLib¬
1085                                                    rmG_symbolcastName)
1086          end if
1087          if xSprite <> 0 then   -- 0 if newMessage couldn't get one
1088            placeSymbolInText(sprite xSprite, index)
1089          else -- can't do anything, no more sprites.
1090            alert "No more sprites for symbols!!"
1091          end if
1092
1093          if rmG_mode = #display OR rmG_state = #decIn or rmG_state =
1094  #done then
1095            --can't move symbols
1096            set the moveableSprite of sprite xSprite = FALSE
1097          end if
1098
1099        else nothing   -- code word does not have a symbol
1100      else nothing   -- not a code word
1101    end repeat      -- finished processing symbols
1102
1103
1104    -- Second, place the typein text fields
1105    if rmG_state <> #new then    -- coder or decoder needs to see typedtxt
1106      placeGuesses
1107    end if
1108
1109    updateStage
1110
1111  end placeGraphics
1112
```

```
1113  -- message authoring handlers continued
1114  --- message is a msgBody without state
1115  --- Needs to insert spaces when text is laid out
1116
1117  on layoutText message, cWordColor
1118
1119    global rmG_traceFlag
1120
1121    if rmG_traceFlag then
1122      put "In REBUS:layoutText"
1123      put " coded word color = " & cWordColor
1124    end if
1125
1126    -- text invisible until changed
1127    put "  " into member "MessageSpace"
1128    set the forecolor of member "MessageSpace" = 0
1129
1130    set ts = ""
1131    set indices = []
1132    set symbols = [:]
1133    set charPos = 1
1134    repeat with x in message
1135      set nspaces = 1
1136      set ts = ts & getProp(x, #text)
1137      if getProp(x, #code) then
1138        add(indices, getProp(x, #wordIndex))
1139        addProp(symbols, getProp(x, #wordIndex), getProp(x, #symbol))
1140      end if
1141    end repeat
1142
1143    put ts into field "MessageSpace"
1144
1145    -- color the text and make it visible
1146    repeat with x = 1 to the number of words in ts
1147      if getOne(indices, x) then  -- x is a code word
1148        if getProp(symbols, x) <> 0 then -- x has been coded
1149          set the forecolor of word x of field "MessageSpace" = cWordColor
1150        else
1151          set the forecolor of word x of field "MessageSpace" = 6 --red
1152        end if
1153      else
1154        set the forecolor of word x of field "MessageSpace" = 3 -- blue
1155      end if
1156    end repeat
1157
1158  end layoutText
1159
1160
1161  --- layout index handlers
1162  --- The layout index records information about the position and
1163  --- location of coded words in the "MessageSpace" authoring and
1164  --- display area.
1165  --- It is a property list with the following structure:
```

```
1166  ---
1167  ---  [ layoutIndex: [ firstCharPos, firstCharLoc, ¬
1168                        lastCharPos, lastCharLoc, symbolsprite, msgIndex]
1169  ---    etc.....  ]
1170  --- msgIndex is a pointer to the list location of the word in the
1171  --- rmG_msgBody data structure.
1172
1173
1174  on addIndexedWord  wordIndex
1175    global rmG_layoutIndex
1176
1177    if getOne(getWordIndices(), wordIndex) then
1178      return 0
1179    else addProp(rmG_layoutIndex, wordIndex, [])
1180
1181  end addIndexedWord
1182
1183  ------------------------------------------------------------------------
1184
1185  --- Properties in rmG_layoutIndex should NEVER be set outside of
1186  --- this function!
1187  --- Error handling should be inserted.
1188
1189  on setLayoutProp  wordIndex,  indexProp,  val
1190    global rmG_layoutIndex
1191
1192    if listp(getProp(rmG_layoutIndex, wordIndex)) then   --index is valid
1193      case indexProp of
1194        #firstCharPos:
1195          setAt(getProp(rmG_layoutIndex, wordIndex), 1, val)
1196        #firstCharLoc:
1197          setAt(getProp(rmG_layoutIndex, wordIndex), 2, val)
1198        #lastCharPos:
1199          setAt(getProp(rmG_layoutIndex, wordIndex), 3, val)
1200        #LastCharLoc:
1201          setAt(getProp(rmG_layoutIndex, wordIndex), 4, val)
1202        #symbolSprite:
1203          setAt(getProp(rmG_layoutIndex, wordIndex), 5, val)
1204        #msgIndex:
1205          setAt(getProp(rmG_layoutIndex, wordIndex), 6, val)
1206        otherwise
1207          alert "ERROR: " & indexProp & " not a valid property for ¬
1208                        rmG_layoutIndex"
1209      end case
1210    else
1211      alert "ERROR: Invalid word index. Word " & wordIndex & ¬
1212                    " may not be a coded word."
1213    end if
1214
1215  end setLayoutProp
1216
1217
1218
```

```
1219  --- layout index handlers continued
1220  --- Access function for LayoutIndex
1221
1222  on getLayoutProp wordIndex, indexProp
1223    global rmG_layoutIndex
1224
1225    set val = []
1226    if listp(getProp(rmG_layoutIndex, wordIndex)) then   --index is valid
1227      case indexProp of
1228        #firstCharPos:
1229          set val = getAt(getProp(rmG_layoutIndex, wordIndex), 1)
1230        #firstCharLoc:
1231          set val = getAt(getProp(rmG_layoutIndex, wordIndex), 2)
1232        #lastCharPos:
1233          set val = getAt(getProp(rmG_layoutIndex, wordIndex), 3)
1234        #LastCharLoc:
1235          set val = getAt(getProp(rmG_layoutIndex, wordIndex), 4)
1236        #symbolSprite:
1237          set val = getAt(getProp(rmG_layoutIndex, wordIndex), 5)
1238        #msgIndex:
1239          set val = getAt(getProp(rmG_layoutIndex, wordIndex), 6)
1240        otherwise
1241          alert "ERROR: " & indexProp & " not a valid property."
1242      end case
1243    else
1244      alert "ERROR: Invalid word index. "
1245    end if
1246
1247    return val
1248
1249  end getlayoutProp
1250
1251
1252  on getWordIndices
1253    global rmG_layoutIndex
1254
1255    set indices = []
1256    repeat with i = 1 to count(rmG_layoutIndex)
1257      add(indices, getPropAt(rmG_layoutIndex, i))
1258    end repeat
1259
1260    return indices
1261  end getWordIndices
1262
1263
1264
1265  -- return the wordIndex associated with the sprite
1266
1267  on getSpriteWord spriteNum
1268
1269    set codeWords = getWordIndices()
1270    repeat with i in codeWords
1271      if getLayoutProp(i, #symbolSprite) = spriteNum then return(i)
```

```
1272      end repeat
1273
1274      return(0)  -- no coded wrod associated with sprite
1275
1276    end getSpriteWord
1277
1278
1279    --- layout index handlers continued
1280    ------------------------------------------------------------
1281    -- Should only be called after the template has been laid out
1282    -- in the MessageSpace.  Otherwise the information will not be
1283    -- correct.
1284
1285    on makeLayoutIndex msgWords
1286      global rmG_layoutIndex
1287
1288      -- for safety make sure LayoutIndex is clear before starting
1289      set rmG_layoutIndex = [:]
1290      set textOrigin = the loc of sprite 2  --sprite for MessageSpace"
1291
1292      set i = 1
1293      repeat with x in msgWords
1294        set index = getProp(x, #wordIndex)
1295        addIndexedWord(index)
1296        set startPos = getProp(x, #charPos)
1297        setLayoutProp(index, #firstCharPos, startPos)
1298        setLayoutProp(index, #firstCharLoc, ¬
1299                  myCharPosToLoc(member "MessageSpace", startPos) )
1300
1301        set endPos = startPos + length(getProp(x, #text)) - 1
1302        setLayoutProp(index, #lastCharPos, endPos)
1303        setLayoutProp(index, #lastCharLoc, ¬
1304                   myCharPosToLoc(member "MessageSpace", endPos))
1305        setLayoutProp(index, #msgIndex, i)
1306        set i = i+1
1307
1308        -- assume that if the index has not been created,
1309        -- sprites have not yet be assigned to symbol graphics
1310        -- Could use this property only for coded words but it is probably
1311        -- not worth it.
1312        setLayoutProp(index, #symbolSprite, 0)
1313
1314      end repeat
1315
1316    end makeLayoutIndex
1317
1318
1319
1320
1321    on clearLayoutSymbols
1322      global rmG_layoutIndex
1323
1324      repeat with x in rmG_layoutIndex
```

```
1325        setAt(x, 5, 0)
1326      end repeat
1327
1328    end clearLayoutSymbols
1329
1330
1331
1332    -- View Handlers    Sprites 39 to 44 are reserved for guesses
1333
1334    on HideGuesses
1335      repeat with i = 39 to 44
1336        puppetSprite i, TRUE
1337        set the visible of sprite i = FALSE
1338      end repeat
1339    end HideGuesses
1340
1341    on ShowGuesses
1342      repeat with i = 39 to 44
1343        puppetSprite i, TRUE
1344        set the visible of sprite i = TRUE
1345      end repeat
1346    end ShowGuesses
1347
1348    on HideSymbols
1349      global rmG_state, rmG_userName
1350
1351      if rmG_state = #decIn then
1352        if getProp(rmG_mailData, #to) = rmG_userName then
1353          --- from the decoder's inbox
1354          set wcolor = 0 -- coded words white
1355        else -- User sent this message to someone else
1356          set wcolor = 6 -- coded words red
1357        end if
1358      else
1359        if getProp(rmG_mailData, #to) <> rmG_userName then
1360          --- User is not the coder
1361          set wcolor = 0 -- coded words white
1362        else -- user is the coder
1363          set wcolor = 6 -- coded words red
1364        end if
1365      end if
1366
1367      set codeWords = getWordIndices()
1368      repeat with i in codeWords
1369        set sNum = getLayoutProp(i, #symbolSprite)
1370        if sNum then
1371          set the visible of sprite sNum = FALSE
1372          set the forecolor of word i of field "MessageSpace" = wcolor
1373        end if
1374      end repeat
1375
1376    end HideSymbols
1377
```

```
1378  on ShowSymbols
1379
1380    set codeWords = getWordIndices()
1381    repeat with i in codeWords
1382      set sNum = getLayoutProp(i, #symbolSprite)
1383      if sNum then
1384        set the visible of sprite sNum = TRUE
1385        set the forecolor of word i of member "MessageSpace" = 0
1386      end if
1387
1388    end repeat
1389
1390  end ShowSymbols
1391
1392  -- Type-in text fields used to record decoder's solution for the
1393  message.
1394  -- Eventually these should be changed to list boxes for younger
1395  children.
1396
1397  -- Assume sprites 39-44 have been reserved for guesses
1398  -- Field cast members "Guess1" through "Guess6" are reserved for
1399  guesses.
1400  -- The width of these cast members should be set at 100 pixels
1401  -- or less (currently 90). This cannot be done with Lingo.
1402  -- The boxtype should be #fixed.  If text can't fit, the boxtype
1403  -- should be changed to #scroll at runtime.
1404
1405  ------------------------------------------------------
1406  -- assumes that the global variable rmG_msgBody has been
1407  -- initialized.
1408
1409  on initializeGuesses
1410    global rmG_msgBody, rmG_guesses
1411
1412    --- for safety reset rmG_guesses
1413    set rmG_guesses = []
1414    if not listp(rmG_msgBody) or count(rmG_msgBody) = 0 then
1415      alert "ERROR: rmG_msgBody not initialized correctly"
1416      return
1417    end if
1418
1419    set gNum = 1
1420    repeat with x in getAt(rmG_msgBody, 2)
1421
1422      if getProp(x, #code) then  --this word needs a guess
1423
1424        set cNum = the number of member ("Guess" & gNum)
1425        put getProp(x, #guess) into field cNum
1426
1427        -- format fields
1428        -- most text formatting is done in startMovie handler
1429        set the forecolor of member cNum = 6  -- red
1430        set the boxtype of member cNum = #adjust
```

```
1431        set the border of member cNum = 1
1432
1433        -- assign a sprite (from 39-44) to this guess
1434        set sNum = 38 + gNum     -- first guess is 1
1435
1436        -- Although macromedia says you don't need to
1437        -- puppet sprites in Director 6,
1438        -- the guesses do not appear correctly unless the
1439        -- sprites are puppeted.  This is probably a result
1440        -- of the fact that there is nothing in the sprite
1441        -- channels for guesses in the score.
1442        -- In any event unless this method turns out to be
1443        -- unstable.  We will use it.
1444        puppetsprite sNum, TRUE
1445
1446        set the visible of sprite sNum = FALSE
1447        set the moveablesprite of sprite sNum = FALSE
1448        set the loc of sprite sNum = point(-100, -100)  --offstage
1449        set the memberNum of sprite sNum = cNum
1450        -- set the editable of sprite sNum = FALSE
1451
1452        -- record guess info into rmG_guesses
1453        set guess = [:]
1454        addProp(guess, #cast, cNum )
1455        addProp(guess, #sprite, sNum)
1456        addProp(guess, #wordIndex, getProp(x, #wordIndex))
1457        append(rmG_guesses, guess)
1458
1459        set gNum = gNum + 1
1460      else nothing   --- it is not a coded word
1461    end repeat
1462
1463  end initializeGuesses
1464
1465
1466  on cleanupGuesses
1467    global rmG_guesses
1468
1469    repeat with x in rmG_guesses
1470      set sNum = getProp(x, #sprite)
1471      set the visible of sprite sNum = FALSE
1472      set the memberNum of sprite sNum = 0
1473      put " " into member getProp(x, #cast)
1474    end repeat
1475
1476    set rmG_guesses = []
1477
1478  end cleanupGuesses
1479
1480
1481  -- Places a sprite for each guess just below the word that it
1482  corresponds to.
```

```
1483  -- Assume that rmG_guesses has been properly initialized with each
1484  guess.
1485
1486  on placeGuesses
1487    global rmG_guesses, rmG_state, rmG_mode
1488
1489    --- sprite 2 is Rebus message field. It's cast member is
1490  "MessageSpace"
1491    set textOrigin = the loc of sprite 2
1492
1493    repeat with x in rmG_guesses
1494
1495      if getLayoutProp(getProp(x, #wordIndex), #symbolSprite) then
1496        -- this word is coded;  it needs a guessbox
1497
1498        set guessSprite = getProp(x, #sprite)
1499        set guessMember = the memberNum of sprite guessSprite
1500
1501        if rmG_state = #decIn and rmG_mode = #author then
1502          set the editable of member guessMember = TRUE
1503        else set the editable of member guessMember = FALSE
1504
1505        -- place the sprite
1506        set the loc of sprite guessSprite = ¬
1507              getLayoutProp(getProp(x, #wordIndex), ¬
1508                            #firstCharLoc) + textOrigin + point(0, 2)
1509        set the visible of sprite guessSprite = TRUE
1510
1511      end if   -- x is a coded word
1512
1513    end repeat  -- x in rmG_guesses
1514    updatestage
1515
1516  end placeGuesses
1517
1518  ------------------------------------------------------------
1519  -- Reads latest guesses from the guess cast members back into
1520  -- the message body.
1521
1522  on putGuessesInMsg
1523    global rmG_msgBody, rmG_guesses
1524
1525    set message = getAt(rmG_msgBody, 2)
1526
1527    repeat with x in rmG_guesses
1528      set str = the text of member getProp(x, #cast)
1529      repeat with y in message
1530        if getProp(y, #code) then
1531          if getProp(y, #wordIndex) = getProp(x, #wordIndex) then
1532            setProp(y, #guess, str)
1533            exit repeat
1534          end if
1535        end if
```

```
1536
1537        end repeat
1538      end repeat
1539
1540    end putGuessesInMsg
1541
1542
1543
1544
1545    -- this function is obsoleted..
1546
1547    on sendIt
1548      global rmG_msgBody, rmG_mailData, rmG_state, rmG_mode
1549      global rmG_noSimulate
1550
1551      -- It seems that rmG_state gets reset on return to EmailMain
1552      -- reset it here
1553      -- set rmG_state = getAt(rmG_msgBody, 1)
1554      -- IF SIMULATE SEND but don't change MsgBody, otherwise send
1555
1556      if not rmG_noSimulate then
1557        simulateSend()
1558
1559      else   -- called from email main
1560
1561        -- next is not sufficient; should check for a valid address
1562        if the text of member "To" = "" then
1563          alert "No one to send to...."
1564          return
1565        end if
1566
1567        -- read header information from fields
1568        -- back into the message body
1569
1570        updateHeader
1571
1572        toggleRebusState()
1573        set rmG_mode = #display  -- can't edit message further
1574
1575        putGuessesInMsg
1576
1577        --  ADD UPDATED MESSAGE BODY TO rmG_mailData
1578
1579        setaProp rmG_mailData, #msgbody, rmG_msgBody
1580
1581
1582        --  SEND MESSAGE TO EMAIL MAIN
1583        ---  NOTE: sendToggle lets messageHandler know to send the message
1584        ---  rather than simply save a previously sent message
1585
1586        set sendToggle = 1
1587
1588        tell the stage
```

```
1589
1590        global rmG_mailData, rmG_mode
1591
1592        messageHandler sendToggle
1593
1594      end tell
1595
1596
1597    end if
1598
1599  end
1600
1601
1602  on simulateSend
1603    global rmG_msgBody, rmG_mailData, rmG_state
1604
1605    alertSent()
1606    toggleRebusState()
1607    setProp(rmG_mailData, #From, the text of field "to")
1608    clearMessageSpace   -- this clears symbols and guesses
1609    openMessage
1610
1611  end simulateSend
1612
1613  ------------------------------------------------------------------------
1614
1615  on toggleRebusState
1616    global rmG_state, rmG_msgBody
1617
1618    case rmG_state of
1619      #new:
1620        deleteAt(rmG_msgBody, 1)
1621        AddAt(rmG_msgBody,1,#decIn)
1622      #decIn:
1623        deleteAt(rmG_msgBody, 1)
1624        AddAt(rmG_msgBody,1,#codIn)
1625      #codIn:
1626        deleteAt(rmG_msgBody, 1)
1627        addAt(rmG_msgBody, 1, #decIn)
1628      #gotIt:
1629        -- coder indicated that they solved it before sending
1630        -- deleteAt(rmG_msgBody, 1)
1631        -- addAt(rmG_msgBody, 1, #gotIt)
1632    end case
1633
1634    set rmG_state = getAt(rmG_msgBody, 1)
1635
1636  end toggleRebusState
1637
1638
1639  -- Text format
1640  ------------------------------------------------------------
1641  -- Initialize formatting of all visible text fields
```

```
1642  -- Should be called when movie starts
1643
1644  on formatFields
1645
1646    repeat with i = 1 to 6
1647      SetTextInfo "Guess" & i, "  ", "left", "arial", 20, "bold"
1648    end repeat
1649
1650    SetTextInfo "To", " ", "left", "arial", 14, "bold"
1651    SetTextInfo "ToDown", "  ", "left", "arial", 14, "bold"
1652    set the lineHeight of field "ToDown" = 30
1653    -- set the lineHeight of field "To" = 18
1654    --set the border of member "To" = 1
1655    set the border of member "ToDown" = 1
1656    set the margin of member "To" to 4
1657    set the margin of member "ToDown" to 8
1658
1659  end formatFields
1660
1661
1662  ------------------------------------------------------
1663  -- SetUpMessageSpace
1664  -- need to know difference between default line height
1665  -- for a given fontsize and the lineHeight we have set
1666
1667  on setupMessageSpace
1668    global rmG_spaceWidth      -- pixel width of a space in "MessageSpace"
1669
1670    setTextInfo "MessageSpace", "  ", "left", "arial", 32, "bold"
1671
1672    set the forecolor of member "MessageSpace" = 0  --white is invisible
1673
1674    -- standard lineheight for 32 pt font = 39
1675    set the lineHeight of member "MessageSpace" = 95
1676
1677    set rmG_spaceWidth = charwidth(1, "MessageSpace")
1678
1679
1680  end setupMessageSpace
1681
1682
1683
1684  --  Utilities
1685  ------------------------------------------------------------
1686  --- MyCharPosToLoc
1687  --- Adjusts for bug in Lingo charPosToLoc function and returns
1688  --- correct results regardless of whether the lineHeight of the
1689  --- field has been set.
1690
1691  --- Lingo bug causes charPosToLoc function to return different
1692  --- results if the lineheight of a field has been set - even
1693  --- if the lineheight is set to exactly the value it started at
1694  --- The function works correctly as long as lineheight has not
```

```
--- been set.  If the lineheight has been set, vertical
--- coordinates = point(x,-2) for characters in the first line
--- and point(x, line#*lineHeight -2) characters on subsequent
--- lines.  Unfortunately, this is not where the characters are!

on myCharPosToLoc  fieldMember,  charPos

  set maybeLoc = charPosToLoc(fieldMember, 1)
  if getAt(maybeLoc, 2) <> -2 then  -- Macromedia is good
    return(charPosToLoc(fieldMember, charPos))

  else  -- fix the loc

    set memNum = the number of member fieldMember
    set fHeight = getFontHeight(memNum)
    set lHeight = the lineHeight of field memNum  --if badloc, this is
correct
    if the fontSize of field memNum <=24 then
      set topHeight = 2
    else set topHeight = 6
    set belowHeight = lHeight - fHeight - 2

    --- adjust Lingo value to be correct
    set badLoc = charPosToLoc(fieldMember, charPos)
    return( badLoc + point(0, fHeight +2 + topHeight))
  end if

end myCharPosToLoc

-----------------------------------------------------------------------

--- GetFontHeight is used to determine the vertical distance of
--- tallest character for any font.
--- This distance excludes any space above or below the font.

on getFontHeight  fieldMemberNum
  --- lineheight may have been changed so need to create a
  --- new cast member with font, fontsize and style. To determine
  --- the fontHeight

  set tmpNum = findEmpty(member 1)
  set tmpMember = new(#field, member tmpNum of castLib "Internal")

  put "Test" into field tmpNum
  set the font of field tmpNum  = the font of field fieldMemberNum
  set the fontSize of field tmpNum = the fontSize of field
fieldMemberNum
  set the fontStyle of field tmpNum = the fontStyle of field
fieldMemberNum

  -- get the location of the lower left corner of 1st char
  set bottomLeft = charPosToLoc(member tmpNum, 1)

  set tmpMember = 0    -- clear reference to the field before erasing
```

```
1748      erase member tmpNum
1749
1750      if the fontsize of field fieldMemberNum <=24 then
1751        -- 1st line starts 2 points below top of field
1752        return(getAt(bottomLeft,2) - 2)
1753      else
1754        -- 1st line starts 6 points below top of field
1755        return(getAt(bottomLeft,2) - 6)
1756      end if
1757
1758    end getFontHeight .
1759
1760    ------------------------------------------------------------
1761    --- LocToWordPos returns the index of the word under loc in
1762    --- the field. Vertical space in the field is assigned to the
1763    --- closest char in the field. If loc is on a space between
1764    --- words (horizontal only, see below), the function returns 0.
1765    --- If the loc is not in the field, results are unpredictable.
1766    --- The Lingo loctoCharPos function will return either the first
1767    --- or the last character in the field, depending upon the loc.
1768    --- Comparable to the built-in Lingo locToCharPos function,
1769    --- the location argument is assumed to be relative to the origin
1770    --- of the text field.
1771
1772    --- Uses the Lingo function locToCharPos.  locToCharPos is not
1773    --- subject to the Lingo lineHeight bug.  All locations with
1774    --- vertical coordinates within a lineheight return a character
1775    --- on that line.  For example, if lineheight is 36 then any location
1776    --- in the interval, [point(x,0) point(x, 36)] with return a character
1777    --- on the first line, any location in the interval,
1778    --- [point(x, 37) point(x, 72)] will return a character in the second
1779    --- line, etc. Note: a character is returned even if the location
1780    --- is on the white space between lines.
1781
1782    on locToWordPos  fieldMember,  locInField
1783
1784      set charPos = locToCharPos(fieldMember, locInField)
1785
1786      -- FIX THIS TO CHECK THAT LOC IS IN FIELD ELSE RETURN 0.
1787      -- locTocharPos will return 1 if the loc is above or left of the field
1788      -- It will return lenght(text) + 1 if the loc is below or right of
1789    field
1790
1791      if char charPos of field fieldMember = " " then -- on a space
1792        return(0)
1793      else
1794        return(the number of words in char 1 to charPos of field
1795    fieldMember)
1796      end if
1797
1798    end locToWordPos
1799
1800    ------------------------------------------------------------
```

```
1801
1802  on charWidth charPos, afield
1803    return GetAt(charPosToLoc(member afield, charPos+1) - ¬
1804           charPosToLoc(member afield, charPos), 1)
1805  end charWidth
1806
1807  ------------------------------------------------------------------------
1808  -- determines how much horizontal space in pixels is taken
1809
1810  on wordSpaceH wordIndex
1811
1812    set textOrigin = the loc of sprite 2
1813    set lastC = length(word 1 to (wordIndex+1) of field "MessageSpace")¬
1814                - length(word wordIndex +1 of field "MessageSpace") + 1
1815    if wordIndex = 1 then
1816      set firstC = 1
1817    else
1818      set firstC = length(word 1 to (wordIndex-1) of field "MessageSpace")
1819  + 1
1820    end if
1821
1822    set firstLoc = myCharPosToLoc(member "MessageSpace", firstC)
1823    set lastLoc =  myCharPosToLoc(member "MessageSpace", lastC)
1824
1825    if getAt(firstLoc, 2) <> ¬
1826                getAt(getLayoutProp(wordIndex, #firstCharLoc), 2) then
1827      -- if word before is not on same line this is first in line
1828
1829      set wspace = getAt(lastLoc,1) - the left of sprite 8
1830
1831    else if getAt(lastLoc, 2) <> ¬
1832                getAt(getLayoutProp(wordIndex, #lastCharLoc), 2) then
1833      -- if word after is not on same line this is last in line
1834
1835      set wspace = the right of sprite 8 - getAt(firstLoc,1)
1836
1837    else --- this is in the middle of the line
1838
1839      set wspace = getAt(lastloc, 1) - getAt(firstLoc, 1)
1840
1841    end if
1842
1843    return wspace
1844  end wordSpaceH
1845
1846
1847
1848  --- centerInBox sets sprite location to align registration point
1849  --- to the center of the rectangle
1850
1851  on centerInBox aSprite, aRect
1852
```

```
1853    set wordHCenter = getAt(aRect,1) + integer((getAt(aRect,3)-
1854  getAt(aRect,1))/2)
1855    set wordVCenter = getAt(aRect,2) + integer((getAt(aRect,4)-
1856  getAt(aRect,2))/2)
1857    set the locH of sprite aSprite = wordHCenter
1858    set the locV of sprite aSprite = wordVCenter
1859
1860    return(the loc of sprite aSprite)
1861  end centerInBox
1862
1863  ------------------------------------------------------------------------
1864  --- CenterOnWord returns a point that is the center of the
1865  --- word contained in the space between startLoc and endLoc.
1866  --- Mainly it takes care of vertical position of word center.
1867  --- It centers at 1/2 the height of the font. (Not lineheight)
1868  --- Assumes startloc and endloc are bottom left corners of first
1869  --- and last characters.
1870
1871  on centerOnWord  fieldMember,  startLoc,  endLoc
1872
1873    -- for performance make this a global for "MessageSpace"
1874    set fHeight = getFontHeight(fieldMember)
1875    set xCoord = getAt(startLoc, 1)+ ((getAt(endLoc, 1) - getAt(startLoc,
1876  1))/2)
1877    return(point(xCoord, getAt(startLoc, 2) - fHeight/2))
1878
1879
1880  end centerOnWord
1881
1882  ------------------------------------------------------------------------
1883
1884  -- makeWordRect returns a rect that bounds a word in a field member.
1885  -- The rect includes 1/2 of the space below the line of the word.
1886  -- The return value has coordinates relative to the field Member.
1887  -- Assume startloc and endloc are coordinates relative to origin
1888  -- of the fieldMember argument.
1889  -- Assume that startloc and endloc point to the bottom left corner
1890  -- of the first and last characters of the word.
1891
1892  on makeWordRect  fieldMember,  startloc,  endloc
1893
1894    set wordRect = rect(0,0,0,0)   -- value to be returned
1895
1896
1897    set fieldNum = the number of fieldMember
1898
1899
1900  end makeWordRect
1901
1902
1903  ------------------------------------------------------------------------
1904
1905  on shiftTextRight  cPos,  pixDistance,  fieldNum,  spaceWidth
```

```
1906
1907    set nchars = integer( float(pixdistance) / spaceWidth)
1908    repeat with i = 1 to nchars
1909      put " " before char cPos of field fieldNum
1910    end repeat
1911    return nchars
1912
1913  end shiftTextRight
1914
1915
1916  ------------------------------------------------------------------------
1917  --- codeWordP returns true if the word is a word
1918  --- in the message to be coded
1919  ---
1920
1921  on codeWordP  wordIndex
1922    global rmG_msgBody
1923
1924    set message = getAt(rmG_msgBody,2)
1925    repeat with i = 1 to count(message)
1926      set nextItem = getAt(message,i)
1927      if getProp(nextItem, #code) then
1928        if getProp(nextItem, #wordIndex) = wordIndex then
1929          return(1)
1930        end if
1931      end if
1932    end repeat
1933
1934    return(0)
1935
1936  end codeWordP
1937
1938
1939
1940  on fillToList
1941    global rmG_registeredUsers
1942
1943    put "" into field "ToDown"
1944    repeat with uname in rmG_registeredUsers
1945      put uname & RETURN after field "ToDown"
1946    end repeat
1947
1948  end filltoList
1949
1950
1951
1952  on alertSent
1953
1954    go to frame "sent"
1955
1956  end alertSent
1957
1958  ------------------------------------------------------------------------
```

```
1959
1960  on flashSprite spriteNum
1961    set the visible of sprite spriteNum = ¬
1962                not (the visible of sprite spritenum)
1963
1964    starttimer
1965    repeat while the timer < 30
1966      nothing
1967    end repeat
1968
1969  end flashSprite
1970
1971
1972
1973  -- programming utility to easily copy scripts to cast members
1974  -- call from the message window
1975
1976  on copyScript fromCast, toCast1, toCast2
1977    repeat with i = toCast1 to toCast2
1978      set the scriptText of member i = the scripttext of member fromCast
1979    end repeat
1980  end copyScript
1981
1982
1983  ------------------------------------------------------------------------
1984
1985  on clearScripts fromCast, toCast
1986    repeat with i = fromCast to toCast
1987      set the scriptText of member i = ""
1988    end repeat
1989  end clearScripts
1990
1991
1992
1993  -- score scripts sentmsg_loop
1994
1995  on exitFrame
1996    flashSprite(49)
1997    go to the frame
1998  end
1999
2000
2001  on mouseDown
2002
2003    -- make sure the flashing sprite is visible
2004    set the visible of sprite 49 = TRUE
2005    go to frame 2
2006
2007  end
2008
2009  ------------------------------------------------------------------------
2010
2011  -- score scripts fr_startMenus
```

```
2012
2013  on exitFrame
2014    global rmG_testState
2015
2016    -- first clear any existing menus
2017    installMenu 0
2018
2019    -- maybe should setup graphics palette here
2020    if rmG_testState then
2021      installMenu "userTestMenu"
2022    else
2023      installMenu "standardMenu"
2024    end if
2025
2026  end
2027
2028  -- score scripts fr_gotIt_loop
2029
2030  on exitFrame
2031
2032    repeat with i = 21 to 35
2033      togglevisible sprite i
2034    end repeat
2035
2036    go to the frame
2037  end
2038
2039
2040
2041  -- API Public Handlers
2042  ------------------------------------------------
2043  --- Ugly hack to work around problem with Director startup
2044  --- of MIAWs.  The problem is that, after calling a handler in the
2045  --- MIAW, the StartMovie handler for the MIAW does not run until
2046  --- the calling movie advances to its next frame.
2047  --- Therefore, the calling sequence in the calling movie
2048  --- has to be engineered so that the real handlers in the MIAW do not
2049  --- run until after control has been transfered back to the calling
2050  --- movie. However, at least one handler in the MIAW must be called
2051  --- by the calling movie before the StartMovie handler will run.
2052
2053  --- startMeUp is the fake handler that, when called by the
2054  --- main movie, will upon return to the main movie,
2055  --- cause this movie's startMovie handler to run.
2056
2057  --- The second part of this wormy hack is contained in the MIAW's
2058  --- startMovie handler... It is a call to a workAround handler in
2059  --- the calling movie called continueComponent
2060  --- The calling movie's continueRebus handler calls the real handlers
2061  --- in the MIAW.
2062
2063  on emc_startMeUp
2064    -- put "Macromedia sucks!"
```

```
2065    global rmG_noSimulate
2066    set rmG_noSimulate = TRUE
2067    return(1)
2068
2069  end emc_startMeUp
2070
2071  ------------------------------------------------------------------------
2072  -- API Public Handlers continued
2073  -- initWindow is called by the stage when a Rebus movie is opened
2074  -- It sets up the user information
2075  -- Need to wait for the message before the templates and symbol group
2076  -- can be setup.
2077
2078  on emc_initWindow userName
2079    global rmG_registeredUsers      -- list of KidCode system usernames
2080    global rmG_userName             -- current username
2081    global rmG_UserGroup           -- user group of current user
2082    global rmG_layoutIndex            -- ds to improve efficiency of layout
2083    global rmG_templates
2084    global rmG_traceFlag
2085
2086    if rmG_traceFlag then
2087      put "In REBUS:emc_InitWindow"
2088      put "      userName = " & userName
2089    end if
2090
2091    tell the stage to emh_getRegisteredUsers()
2092    set rmG_registeredUsers = the result
2093    set rmG_userName = userName
2094
2095    tell the stage to emh_getUserData(userName)
2096    set userData = the result
2097    set rmG_UserGroup = getAt(userData, 3)
2098
2099    -- NEXT EXISTS SO THAT REBUS MOVIE CAN BE RUN IN SIMULATION MODE
2100    -- MUST BE SET TO FALSE TO RUN AS EMAIL COMPONENT!!!!
2101    global rmG_noSimulate      -- if TRUE, runs independent of email
2102
2103    If the result = 0 then
2104      set rmG_noSimulate = FALSE
2105    else
2106      set rmG_noSimulate = TRUE
2107    end if
2108
2109
2110    initializeTemplates()  --set up rmG_templates
2111
2112    --- Format the font properties of text fields and the MessageSpace
2113    setUpMessageSpace()
2114    formatFields()
2115
2116    --- allow To field to be a listbox
2117    puppetSprite 50, TRUE
```

```
2118
2119    set rmG_layoutIndex = [:]  -- records word position and loc info by
2120  index
2121
2122    return(1)
2123  end emc_initWindow
2124
2125
2126  -- API Public Handlers continued
2127  --- closeWindow is not called unless Rebus plays as
2128  --- a MIAW.
2129
2130  on emc_closeWindow
2131    finishMovie
2132    return(1)
2133  end emc_closeWindow
2134
2135  ------------------------------------------------------------------------
2136
2137  on emc_getComponentInfo
2138
2139    -- eventually the MIMEtype field will be application/x-rebus
2140
2141    return(list("Rebus",3,#msgHandler,"rebus"))
2142
2143  end emc_getComponentInfo
2144
2145  ------------------------------------------------------------------------
2146
2147  on msh_openMessage mailData, mode
2148    global rmG_traceFlag
2149    global rmG_registeredUsers
2150    global rmG_msgBody, rmG_mailData, rmG_mode, rmG_state, rmG_symbolGroup
2151    global rmG_lineHeightFix
2152    global rmG_noSimulate      --- False to run simulation independent of
2153  email
2154    global rmG_testState
2155    global rmG_templates, gTemplateIndex
2156    global rmG_symbolcastName, rmG_symbolGroup
2157    global rmG_userName
2158
2159    set rmG_mailData = mailData
2160
2161    if rmG_traceFlag then
2162      put "In REBUS:msh_openMessage"
2163      put "      mode = " & mode
2164      put "      maildata = " & mailData
2165    end if
2166
2167    set rmG_mode = mode
2168
2169    if rmG_noSimulate then
2170      set rmG_msgBody = getProp(mailData, #msgbody)
```

```
2171
2172      if count(rmG_msgBody) = 0 then   -- start a new message
2173
2174        startMessage("default")
2175        --- Fill header fields
2176        put getProp(mailData, #to) into field "To"
2177        put rmG_userName into field "From"
2178        put the abbreviated date into field "Date"
2179        put "Rebus Challenge" into field "Re"
2180
2181        -- setup templates and symbols
2182        set rmG_theTemplateIndex = getPos(rmG_templates, "default")
2183        set rmG_symbolGroup = getTemplateProp("default", #symbols)
2184
2185        -- administrator uses a different symbol library
2186        if rmG_testState = 0 then
2187          set rmG_symbolcastName = "Symbols"
2188        else
2189          set rmG_symbolcastName = getTemplateProp("default", #library)
2190        end if
2191
2192      else  --- Handle existing message
2193
2194        set rmG_state = getAt(rmG_msgBody, 1)
2195        set whichTemplate = getAt(rmG_msgBody,3)
2196
2197        clearMessageSpace
2198        putHdrFields()   -- read headers into field members
2199
2200        --- setup symbols in graphics palette to match template
2201        set theTemplate = getAt(rmG_msgBody,3)
2202        set rmG_theTemplateIndex = GetPos(rmG_templates, theTemplate)
2203        set rmG_symbolcastName = getTemplateProp(theTemplate, #library)
2204        set rmG_symbolGroup = getTemplateProp(theTemplate, #symbols)
2205        setupSymbolPalette rmG_symbolcastName
2206        initializeGuesses    -- uses global var rmG_msgBody
2207
2208        case rmG_state of
2209          #new:
2210            layoutText(getat(rmG_msgBody,2), 6)  -- coded words red
2211            makeLayoutIndex(getAt(rmG_msgBody, 2))
2212            placeGraphics()
2213
2214          #decIn:
2215            if getProp(rmG_mailData, #to) = rmG_userName then
2216              --- from the decoder's inbox
2217              layoutText(getat(rmG_msgBody,2), 0) -- coded words white
2218            else -- User sent this message to someone else
2219              layoutText(getat(rmG_msgBody,2), 6) -- coded words red
2220            end if
2221            makeLayoutIndex(getAt(rmG_msgBody, 2))
2222            placeGraphics()
2223
```

```
2224          #codIn:
2225            if getProp(rmG_mailData, #to) <> rmG_userName then
2226              --- User is not the coder
2227              layoutText(getat(rmG_msgBody,2), 0) -- coded words white
2228            else -- user is the coder
2229              layoutText(getat(rmG_msgBody,2), 6) -- coded words red
2230            end if
2231            makeLayoutIndex(getAt(rmG_msgBody, 2))
2232            placeGraphics()
2233
2234          #gotIt:
2235            -- hide scrolling symbol palette
2236            repeat with i = 15 to 20
2237              -- puppetsprite i, TRUE
2238              set the visible of sprite i = FALSE
2239            end repeat
2240            layoutText(getat(rmG_msgBody,2), 6)  -- coded words white
2241            makeLayoutIndex(getAt(rmG_msgBody, 2))
2242            placeGraphics()
2243            go to "gotIt"
2244        end case
2245
2246      end if
2247
2248
2249    else   -- Simulate mode
2250
2251      setupSymbolPalette rmG_symbolcastName
2252      set rmG_state = getAt(rmG_msgBody, 1)
2253      if rmG_state = #new then      -- user picked a template
2254        set rmG_msgBody = newMessageBody(whichTemplate)
2255        layoutText(getAt(rmG_msgBody, 2), 6)   -- 6 is white
2256        makeLayoutIndex(getAt(rmG_msgBody, 2))
2257        initializeGuesses     -- uses global var rmG_msgBody
2258      else   --- don't want to change message body or index
2259        layoutText(getAt(rmG_msgBody, 2), 0)   -- 0 is white
2260        makeLayoutIndex(getAt(rmG_msgBody, 2))
2261        placeGraphics()
2262      end if
2263    end if   -- Simulate
2264
2265    if mode = #display then
2266      set the editable of member "To" = FALSE
2267    else if mode = #author then
2268      fillToList()
2269
2270    end if
2271
2272    return(1)
2273
2274  end msh_openMessage
2275
2276
```

```
2277
2278  on msh_clearMessage
2279
2280    clearMessageSpace
2281    return(1)
2282  end msh_clearMessage
2283
2284
2285
2286  -- API Public Handlers continued
2287
2288  on msh_sendMessage
2289
2290    global rmG_msgBody, rmG_mailData, rmG_state, rmG_mode, rmG_userName
2291    global rmG_noSimulate
2292
2293
2294    --IF SIMULATE SEND do everything differently
2295
2296    if not rmG_noSimulate then
2297      simulateSend()
2298      return(1)
2299    end if
2300
2301    -- called from email main
2302
2303    -- next is not sufficient; should check for a valid address
2304    if the text of member "To" = "" then
2305      alert "No one to send to...."
2306      return(0)
2307    end if
2308
2309
2310    -- read header information from fields
2311    -- back into the message body
2312
2313    updateHeader
2314
2315    toggleRebusState()
2316    putGuessesInMsg
2317    setaProp rmG_mailData, #msgbody, rmG_msgBody
2318
2319    --  ADD UPDATED MESSAGE BODY TO rmG_mailData
2320    -- create a copy for main because we need new rmG_mailData
2321
2322    set msgToSend = duplicate(rmG_mailData)
2323
2324    --- prepare MessageSpace to start a new message
2325    tell the stage to emh_getMessage(0, "rebus")
2326    set rmG_mailData = the result
2327    startMessage("default")
2328
2329    --- Fill header fields
```

```
2330      put getProp(rmG_mailData, #to) into field "To"
2331      put rmG_userName into field "From"
2332      put getProp(rmG_mailData, #date) into field "Date"
2333      put "Rebus Challenge" into field "Re"
2334
2335
2336      if 0 then --old code
2337        set rmG_mode = #display  -- can't edit message further
2338
2339        putGuessesInMsg
2340
2341        --  ADD UPDATED MESSAGE BODY TO rmG_mailData
2342
2343        setaProp rmG_mailData, #msgbody, rmG_msgBody
2344
2345
2346        if 0 then  --- debugging flags
2347          --- email main messageHandler calls resetStage (in main)
2348          --- which should reset rmG_mailData
2349          --- check here to see what has happened
2350          put "IN SendIt; AFTER stage MessageHandler:"
2351          put "Rebus:rmG_mailData = " & rmG_mailData
2352          put "Rebus:rmG_msgBody = " & rmG_msgBody
2353          put "Rebus:rmG_state = " & rmG_state
2354        end if
2355      end if  -- old code
2356
2357      alertSent()
2358      return(msgToSend)  --in old code this was rmG_mailData
2359
2360    end msh_sendMessage
2361
2362
2363
2364    on  msh_replyMessage
2365
2366      global rmG_mailData, rmG_msgBody, rmG_state, rmG_mode, rmG_userName
2367      global rmG_traceFlag
2368
2369      if rmG_traceFlag then
2370        put "In REBUS:msh_replyMessage"
2371      end if
2372
2373      if rmG_state = #gotIt then   --- do nothing and return
2374        alert "You can't reply, please start a new message."
2375        return(0)
2376      end if
2377
2378      --- otherwise process the reply command
2379      -- first swap the fields and reset the mode
2380      set rmG_mode = #author
2381      setProp(rmG_mailData, #to, getProp(rmG_mailData, #from))
2382      setProp(rmG_mailData, #from, rmG_userName)
```

```
2383     setProp(rmG_mailData, #re, "Re: " & getProp(rmG_mailData, #re))
2384     openMessage()
2385
2386     return(rmG_mailData)
2387
2388   end msh_replyMessage
2389
2390
2391   ----------------------------------------------------------------
2392
2393   on msh_PrintMessage
2394
2395     -- needs to be implmented
2396     return(1)
2397
2398   end msh_PrintMessage
2399
2400   --- script of cast member closeWindow
2401
2402   on mouseUp
2403
2404     --  Close the mailbox window
2405
2406     if soundBusy(1) then sound stop 1
2407
2408     tell the stage to emh_killComponent()
2409
2410   end
2411
2412   ----------------------------------------------------------------
2413
2414   --- script of cast member messageSpace
2415
2416   on mouseUp
2417
2418     set textOrigin = the loc of sprite 2
2419     set upLoc = point(the mouseH, the mouseV) - textOrigin
2420     set myword = loctowordpos(member "MessageSpace", upLoc)
2421     set astr = word myword of field "MessageSpace"
2422
2423     if myword > 0 then
2424       if the forecolor of word myword of field "messageSpace" <> 0 then
2425         speak( astr)
2426       end if
2427     end if
2428
2429   end
2430
2431
2432
2433   --- script of cast member symbolButtonUp
2434
2435   on mouseUp
```

```
2436    -- if here then the symbol button was up
2437
2438    set the memberNum of sprite 3 = member "SymbolButtonDown"
2439    hideSymbols()
2440  end
2441
2442  ------------------------------------------------------------------------
2443
2444  --- script of cast member symbolButtonDown
2445
2446  on mouseUp
2447    --- if here then symbol button was down
2448
2449    set the memberNum of sprite 3 = member "SymbolButtonUp"
2450    showSymbols()
2451  end
2452
2453  ------------------------------------------------------------------------
2454
2455  --- script of cast member guessButtonUp
2456
2457  on mouseUp
2458    -- if here then the guesses button was up
2459
2460    set the memberNum of sprite 4 = member "GuessesButtonDown"
2461    hideGuesses()
2462  end
2463
2464
2465
2466  --- script of cast member guessButtonDown
2467
2468  on mouseUp
2469    -- if here then the guesses button was down
2470
2471    set the memberNum of sprite 4 = member "GuessesButtonUp"
2472    showGuesses()
2473  end
2474
2475
2476
2477  --- script of cast member templateButtonUp
2478
2479  on mouseUp
2480    global rmG_templates, rmG_theTemplateIndex
2481
2482    set nTemplates = count(rmG_templates)
2483    if rmG_theTemplateIndex = nTemplates then
2484      set rmG_theTemplateIndex = 1
2485    else
2486      set rmG_theTemplateIndex = rmG_theTemplateIndex +1
2487    end if
2488
```

```
2489    clearMessageSpace
2490    startMessage(getAt(rmG_templates, rmG_theTemplateIndex))
2491
2492  end
2493
2494
2495
2496  --- script of cast member paletteUpButton
2497
2498  -- scroll up picts button
2499  -- sprites 15 thru 20 are reserved for symbols on
2500  --  scrolling symbol palette
2501
2502  on mouseDown
2503
2504    repeat while the stillDown = TRUE
2505
2506      startTimer  --- control scroll speed
2507
2508      repeat with i = 15 to 20   -- palette sprites
2509        scrollUp sprite i
2510        ---- alternative calling code for behavior
2511        -- set xref = getAt (the scriptInstanceList of sprite i,1)
2512        -- call (#scrollup, xref)
2513      end repeat
2514
2515      -- adjust timing of scroll speed
2516      repeat while the timer < 20
2517        nothing
2518      end repeat
2519
2520      updateStage
2521
2522    end repeat   -- while mouse stilldown
2523
2524  end
2525
2526
2527
2528  --- script of cast member paletteDownButton
2529  -- scroll the symbol palette down
2530
2531  on mouseDown
2532
2533    repeat while the stillDown = TRUE
2534
2535      startTimer  --- control scroll speed
2536
2537      repeat with i = 15 to 20   -- palette sprites
2538        scrolldown sprite i
2539      end repeat
2540
2541      -- adjust timing of scroll speed
```

```
2542        repeat while the timer < 20
2543          nothing
2544        end repeat
2545
2546        updateStage
2547
2548      end repeat   -- while mouse stilldown
2549
2550    end
2551
2552
2553    --- script of cast member To
2554
2555    on mouseUp
2556      global rmG_mode
2557      --  Pull down student field: change field from
2558      --  up (member 50) to down (member 51)
2559
2560      if rmG_mode = #author then  -- allow changes to to field
2561        set the memberNum of sprite 50 to the number of member "ToDown"
2562      end if
2563
2564    end
2565
2566
2567
2568    --- script of cast member ToDown
2569
2570    on mouseUp
2571
2572      --  Put selected user name into up version of student field
2573      --  and switch the
2574      --  field from down to up
2575
2576      put word 1 of line (the mouseLine) of field "ToDown" into field "To"
2577
2578      set the memberNum of sprite 50 to the number of member "To"
2579
2580    end
2581
2582
2583    -- sample script for symbol cast member
2584    -- each symbol cast member needs this script
2585
2586    on mouseDown
2587      global RGdownLoc
2588      set RGdownLoc = the loc of sprite the clickon
2589    end mouseDown
2590
2591    on mouseUp
2592      global RGdownLoc
2593      symbolClickUp(point(the mouseH, the mouseV), RGdownLoc)
2594    end mouseUp
```

```
-- sample data structure for a template
-- the elements of this data structure are:
♦     template sentence
♦     indices of coded words
♦     name of symbol cast member
♦     indices of symbol cast members for default symbol palette

["His mother scolded him when he threw the ball through the
window.",[7,9,10,12], "UTsymbols",
[135,124,125,126,132,127,128,4,5,133,6,7,8,134,9]]
```